

💡 ARTICLES

🎓 COURSES

🚀 GUIDES

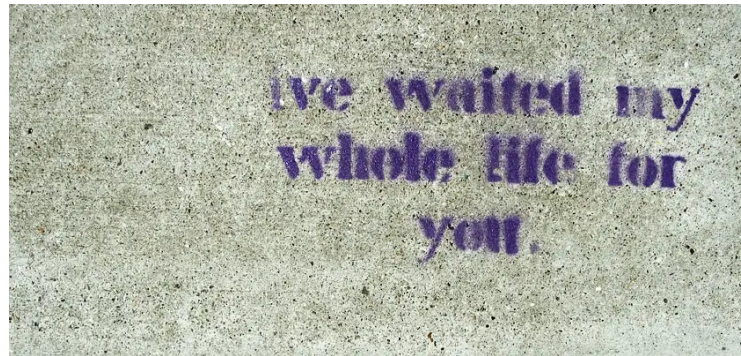
🔗 HOSTING

Search 🔍

👋  
**WE'RE  
SOCIAL!**



## Defer Parsing of JavaScript in WordPress



**DAVID HAYES** / **UPDATED: FEBRUARY 16, 2021**

*Posted In: Back-End Development  
Difficulty: Intermediate*

It's a common complaint when you run your WordPress site through any "page speed score" tool: "defer parsing of JavaScript" and/or "remove render-blocking JavaScript." Today, building on an article Fred first wrote in 2015, I'm going to discuss a way to solve that. It's been possible since WordPress 4.1, which introduced a new filter, `script_loader_tag`. This

## MAIN TOPICS:

- › [Back-End Development](#)
- › [Courses](#)
- › [Editorial](#)
- › [Front-End Development](#)
- › [Quick Guides](#)
- › [Server Administration](#)
- › [Unfiled](#)
- › [Using WordPress](#)

## POPULAR POSTS

- › [Course: Learn WordPress Development](#)
- › [Complete Guide to WordPress Security](#)
- › [WordPress Page Builders, Reviewed](#)
- › [Best Cloud Hosting](#)
- › [Siteground Review](#)
- › [Best WordPress Hosting](#)
- › [How to Start A](#)

filter lets us easily change the HTML markup of enqueued script elements—that is, of JavaScript files that were correctly added into a WordPress site using WordPress’s `wp_enqueue_script` function.

With `script_loader_tag`, we can now easily fix a problem that can significantly impact page speed: lots of render-blocking JavaScript.

## The Problem: Render-Blocking JavaScript

Properly enqueued JavaScript shows up in the head section of your HTML document. On the internet as in nature, the main thing

about a head is that it’s above a body—and this means something fairly serious for site speed, because JavaScript can be *render-blocking*.

---

Long JavaScript files in your head can delay your browser from displaying page content, because its default behavior is first to interpret the JS files themselves.

---

## Blog

### ➤ How to Make A Site

ing” comes from a web  
ult behavior: It wants to  
eive and process  
t’s come higher up in

the page, before it moves any further  
down.

This means that long JavaScript files  
in your head can actually delay your  
browser from displaying the page  
content in the body, because its  
default behavior is first to interpret  
the JS files themselves. In other words,  
JS is *blocking* the browser’s crucial  
function of *rendering* the page out for  
the user to actually see. The result can  
be slow sites and frustrated users.

Google’s **Pagespeed Insights** has been  
pointing out this issue for a while:

**Consider Fixing:**  
Eliminate render-blocking JavaScript and CSS in above-the-fold content

Your page has 18 blocking script resources and 14 blocking CSS resources. This causes a delay in rendering your page.

None of the above-the-fold content on your page could be rendered without waiting for the following resources to load. Try to defer or asynchronously load blocking resources, or inline the critical portions of those resources directly in the HTML.

[Remove render-blocking JavaScript:](#)

- http://wpshout.com/...includes/js/jquery/jquery.js?ver=1.11.3
- http://wpshout.com/...s/jquery/jquery-migrate.min.js?ver=1.2.1
- http://wpshout.com/...shout-js-cookie-demo/cookie.js?ver=4.3.1
- http://wpshout.com/...demo/wpshout-js-cookie-demo.js?ver=4.3.1
- http://wpshout.com/...age/wpshout-no-broken-image.js?ver=4.3.1
- http://wpshout.com/...public/scripts/gdpc-public.js?ver=1.1.23
- http://wpshout.com/...out2014/js/search-box-value.js?ver=4.3.1
- http://wpshout.com/...out2014/js/page-min-height.js?ver=4.3.1
- http://wpshout.com/...ed-plus/scripts/typetris.min.js?ver=4.3.1
- http://wpshout.com/...cator/external-tracking.min.js?ver=6.4.9
- http://wpshout.com/...yntax-highlighter/prism/prism.js?ver=1.0

[Optimize CSS Delivery of the following:](#)

- http://wpshout.com/...ax-highlighter/prism/default.css?ver=1.0
- http://fonts.googleapis.com/css?family=Open+Sans:400,700
- http://wpshout.com/...ntplugins/monarch/css/style.css?ver=1.1
- http://wpshout.com/...ass-demo/wpshout-sass-demo.css?ver=4.3.1
- http://wpshout.com/...teasy-twitter-feed-widget.css?ver=4.3.1
- http://wpshout.com/...p/assets/css/checkbox.min.css?ver=2.3.18
- http://wpshout.com/...lthemes/WPShout2014/style.css?ver=4.3.1



Click to enlarge

However, prior to 4.1 and `script_loader_tag`, the only solution I knew of was to move scripts to the site's footer. Move scripts to the footer is difficult to do by hand. And the plugins I tested that claim to do it automatically didn't work the way I hoped.

Let's move straight to our success story:

## The Goal: Much Less Render-Blocking JS

Here's what we got down to with the solution we'll present below:

**Consider Fixing:**  
Eliminate render-blocking JavaScript and CSS in above-the-fold content

Your page has 3 blocking script resources and 14 blocking CSS resources. This causes a delay in rendering your page.

Approximately 27% of the above-the-fold content on your page could be rendered without waiting for the following resources to load. Try to defer or asynchronously load blocking resources, or inline the critical portions of those resources directly in the HTML.

[Remove render-blocking JavaScript:](#)

- <http://wpshout.com/...includes/js/jquery/jquery.js?ver=1.11.3>
- <http://wpshout.com/...s/jquery/jquery-migrate.min.js?ver=1.2.1>
- <http://wpshout.com/...cator/external-tracking.min.js?ver=6.4.9>

[Optimize CSS Delivery of the following:](#)

- <http://wpshout.com/...ax-highlighter/prism/default.css?ver=1.0>
- <http://fonts.googleapis.com/css?family=Open+Sans:400,700>
- <http://wpshout.com/...nt/plugins/monarch/css/style.css?ver=1.1>
- <http://wpshout.com/...ass-demo/wpshout-sass-demo.css?ver=4.3.1>
- <http://wpshout.com/...t/easy-twitter-feed-widget.css?ver=4.3.1>
- <http://wpshout.com/...p/assets/css/checkboxbox.min.css?ver=2.3.18>
- <http://wpshout.com/...tthemes/WPShout2014/style.css?ver=4.3.1>
- <http://fonts.googleapis.com/...Merrweather:400,400italic,700,700italic>
- <http://fonts.googleapis.com/...lum+Web:400,400italic,700,700italic,900>
- <http://wpshout.com/...ugins/vpiform/css/vpiform.css?ver=4.3.1>
- <http://wpshout.com/...or-vp/assets/css/form.min.css?ver=2.3.18>



Click to enlarge

Now we've only got three JavaScript files that can possibly slow down a page's rendering. The rest are still

there, but they load *in parallel with* the page content, rather than *before* it.

## The Fix: Defer and Async your JavaScript

The first thing to understand is the alternatives to render-blocking JS: `defer` and `async`. We'll explain the difference, but both work similarly: They let the browser load a JS resource “as time permits,” while attending to other things (like page rendering) as well. This means that you *can't* rely on a deferred or asynced JavaScript file being in place prior to page render, as you could without these attributes—but the advantage is that the file won't slow the speed at which the page becomes visible to users.

Those are concepts—now for code.  
(The full code is available [on GitHub](#).)

### 1. GETTING YOUR SCRIPT HANDLES

Every properly enqueued WordPress script has a *handle*: a “nickname” that the site knows to call it by. We're going to need these handles for all scripts,

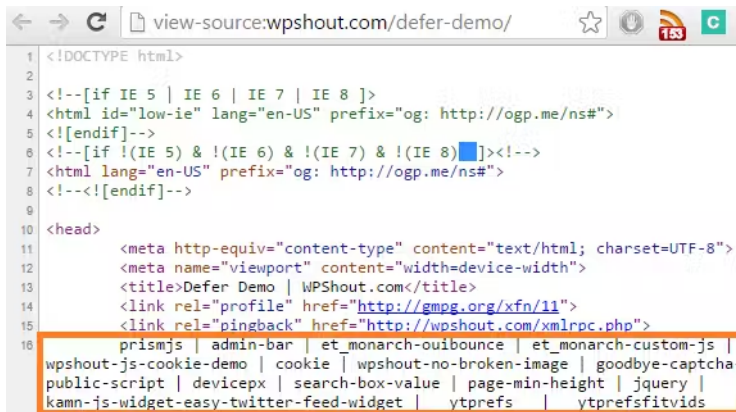
and getting them isn't dead-simple, unfortunately.

It is possible, though:

```
/*
 * Getting script tags
 * Thanks
http://wordpress.stackexchange.
com/questions/54064/how-do-i-
get-the-handle-for-all-
enqueued-scripts
 */

add_action( 'wp_print_scripts',
'wsds_detect_enqueued_scripts'
);
function
wsds_detect_enqueued_scripts()
{
    global $wp_scripts;
    foreach( $wp_scripts->queue
as $handle ) :
        echo $handle . ' | ';
    endforeach;
}
```

This code prints out a list of enqueued handles, separated by | , right into the head of every page:



```
1 <!DOCTYPE html>
2
3 <!--[if IE 5 | IE 6 | IE 7 | IE 8 ]>
4 <html id="low-ie" lang="en-US" prefix="og: http://ogp.me/ns#">
5 <![endif]-->
6 <!--[if !(IE 5) & !(IE 6) & !(IE 7) & !(IE 8)]><!-->
7 <html lang="en-US" prefix="og: http://ogp.me/ns#">
8 <!--<![endif]-->
9
10 <head>
11 <meta http-equiv="content-type" content="text/html; charset=UTF-8">
12 <meta name="viewport" content="width=device-width">
13 <title>Defer Demo | WPShout.com</title>
14 <link rel="profile" href="http://gmpg.org/xfn/11">
15 <link rel="pingback" href="http://wpshout.com/xmlrpc.php">
16 prismjs | admin-bar | et_monarch-ouibounce | et_monarch-custom-js |
wpshout-js-cookie-demo | cookie | wpshout-no-broken-image | goodbye-captcha-
public-script | devicepx | search-box-value | page-min-height | jquery |
kann-js-widget-easy-twitter-feed-widget | ytprefs | ytprefsfritvids
```

You'll only do this once, then use "View Page Source" to copy and paste the handles themselves.

Once you've done this, deactivate this section of the code: we've got our handles, so let's not clog up our head with them anymore. That's why this section is commented out in the code on GitHub—I don't want it to run every time!

## 2. DEFERRING AND ASYNCING RENDER-BLOCKING JAVASCRIPT

We found that we needed to use **defer** and not **async** for WPShout, so I'll walk through the **defer** code. Most of the heavy lifting here is from **an article by Scott Nelle**; thanks, Scott!

```
add_filter(
    'script_loader_tag',
    'wsds_defer_scripts', 10, 3 );
```

```
function wsds_defer_scripts(
    $tag, $handle, $src ) {

    // The handles of the
    // enqueued scripts we want to
    // defer
    $defer_scripts = array(
        'prismjs',
        'admin-bar',
        'et_monarch-ouibounce',
        'et_monarch-custom-js',
        'wpshout-js-cookie-
demo',
        'cookie',
        'wpshout-no-broken-
image',
        'goodbye-captcha-
public-script',
        'devicepx',
        'search-box-value',
        'page-min-height',
        'kamn-js-widget-easy-
twitter-feed-widget',
        '__ytprefs__',
        '__ytprefsfitvids__',
        'jquery-migrate',
        'icegram',
        'disqus',
    );
}
```



```

        if ( in_array( $handle,
$defer_scripts ) ) {
            return '<script src="'
. $src . '" defer="defer"
type="text/javascript">
</script>' . "\n";
        }

    return $tag;
}

```

The `add_filter` line tells us that this code should run anytime an enqueued JavaScript file is about to be printed onto the page as an HTML script element. Letting us filter that HTML is what `script_loader_tag` is for. (If you need an update on filters and WordPress’s hooks system in general, [start here!](#))

The biggest check in this code is doing a single thing defining the `$defer_scripts` array. This array lists out the handles of all the elements we want to defer—the handles we found in step 1. (Your handles will, of course, vary!)

The logic below the array definition (beginning with `if ( in_array( )`

You use **async** when you're linking directly to an *external* JavaScript library. That link would look something like: `<script`  
`src="https://oss.maxcdn.com/libs/re`

`</script>`. Notice how it's a link to the full URL, and the JavaScript will get pulled in

enqueueing *external* JS is a lot less common, at least for us, since most of our enqueued JS is in themes and plugins that host their own code. At any rate, the code for `async` is precisely the same as the code for `defer` —but with the two words switched out. So if you do happen to have a lot of externally hosted enqueued scripts, getting them `async`ed is a very similar technical process to the one we've just covered.

## WHICH SCRIPTS TO DEFER AND ASYNC

You'll notice that we didn't defer everything—a few scripts are still render-blocking. Here are rules of thumb on that:

- › Don't do anything to jQuery. jQuery (handle jQuery) is a key dependency for many other JS files, and you want to let it load early.

- › Any file that's wrapped in a `jQuery( document ).ready( function() { } )` call should be fine to defer. That code basically says “Wait until the entire document object model (DOM) loads,” so racing to get the JavaScript file loaded in the head doesn't serve much purpose.
- › In general, you can defer JavaScript files that rely on user interactions, like clicks and mouse hovers—and files that fix layout details, like center or hide a set of element. Again, these rely on a loaded page to work anyway (which is why they're almost all going to be wrapped in `jQuery( document ).ready( function() { } )`, or else they're liable not to work), so you should be safe to get the page out beforehand.
- › It's, unfortunately, impossible to use this method for JavaScript files that have been added some way other than the generally correct method of enqueueing them. This is another reason to **prefer that method** over other ways of loading

scripts that may appear to work fine at first glance.

## **Want a Simpler Solution? Defer JavaScript Parsing in WordPress with a Plugin**

Because of how common this need to defer JavaScript parsing in WordPress, you can find already-written plugins to do it. The precise methodology then becomes a little less important to you. If you're such an (understandably) hurried person, I've got your back.

A quick note for those hurried people: How well you're able to defer JavaScript parsing with a simple plugin-activation will depend a great deal on the nature of your WordPress setup. If you've got a simple site with a few well-maintained plugins, I'm guessing that simply toggling on a plugin like [Async JavaScript](#) (from the maker of Autoptimize) is likely to work fine. But be aware that these plugin can cause (JavaScript-based) features to break in subtle ways that you might not notice for a while. (This is also true of when you're writing and

maintaining the code we discussed above, but you'll probably do a more thorough test when you've written your code for deferring JavaScript parsing than when it was as easy for you as activating a plugin.)

So in both cases, tread carefully. The makers of these plugins have likely tested extensively, but because WordPress sites are so diverse and varied I'd test a fair amount before rolling with any of these JavaScript-deferring plugins. Here are the three that I found in my research:

- › Async JavaScript (linked above) is by far the most popular, and by the people who've made the famous Autoptimize plugin. (We run Autoptimize here on WPShout)
- › **Speed Booster Pack** — seems a little more more full-featured than the Async JavaScript plugin, and a bit less popular
- › **WP Deferred JavaScript** — this one seems like it could be abandoned, but I'm including it here for completeness

# Two Ways to Defer (or Async) your WordPress JavaScript

At its core we covered a very cool and rather quick way to improve your site speed and user satisfaction. Deferring the parsing of JavaScript can make your WordPress site a great deal faster without any notable impact on reading experience. I hope you now know enough to defer and async your own JavaScript files, and thanks for reading! I'd love to hear comments or questions below.

Image credit: [Celine Nadeau](#)

Don't miss out on the  
very latest 👁👁



**“I love the detailed  
tutorials on  
WPShout... do  
yourself a favor and  
bookmark it!”**

**- Carrie Dils**  
WordPress developer



Get the very latest WordPress  
development tutorials, tricks and  
news in your inbox.

Enter your email address here...

Sign Up Now

## Resources

**ARTICLES**  
**VIDEO QUICK**  
**GUIDES**

## Courses

**WORDPRESS**  
**SECURITY WITH**  
**CONFIDENCE**  
**UP AND**  
**RUNNING WP**

## About Us

**ABOUT &**  
**CONTACT**  
**SPONSOR**  
**WPSHOUT**

## Most Recent

**PHP FOR**  
**BEGINNERS:**  
**STARTING ON**  
**BACKEND**



**BEST  
WORDPRESS  
HOSTING**

**RECOMMENDED  
PRODUCTS**

**FREE WPSHOUT  
COURSES**

**WORDPRESS  
HOSTING GUIDE**



**WORDPRESS  
DEVELOPMENT**

**WORDPRESS  
CUSTOM  
TAXONOMIES:  
HOW AND WHY  
TO CREATE THEM**

**LEARN  
WORDPRESS  
DEVELOPMENT:  
THE BASIC  
COURSE**